Graphes parfaits

Pour S un ensemble, on note $\mathcal{P}_2(S) = \{\{x, y\} : x, y \in S, x \neq y\}$ les sous-ensembles de S de cardinalité 2. Dans tout le sujet, on considère des graphes non-orientés $G = (V, E)$, avec un ensemble fini de sommets V, et un ensemble d'arêtes $E \subseteq \mathcal{P}_2(V)$.
Coloriage. Un coloriage d'un graphe $G = (V, E)$ est une application $V \to \mathbb{N}$. Dans ce contexte, on appelle les entiers des *couleurs*. Un coloriage est *valide* si toute paire de sommets reliés par une même arête a des couleurs différentes. Le *nombre chromatique* de G, noté $\chi(G)$, est le nombre minimal de couleurs nécessaires pour créer un coloriage valide de G.
Sous-graphe induit. Étant donné un graphe $G = (V, E)$ et un sous-ensemble de sommets $W \subseteq V$, le *sous-graphe induit* par W est le graphe $G[W] = (W, E \cap \mathcal{P}_2(W))$. On dit que c'est un sous-graphe induit *propre* si W est inclus strictement dans V.
Cliques. Une *clique* d'un graphe $G = (V, E)$ est un sous-ensemble de sommets $W \subseteq V$ tel que le sous-graphe $G[W]$ induit par W est un graphe complet, c'est-à-dire : $G[W] = (W, \mathcal{P}_2(W))$. On note $\omega(G)$ la cardinalité de la plus grande clique de G.
Anticliques. Une *anticlique* d'un graphe $G = (V, E)$ est un sous-ensemble de sommets $W \subseteq V$ tel que le sous-graphe $G[W]$ induit par W ne contient pas d'arête, c'est-à-dire : $G[W] = (W, \varnothing)$. On note $\alpha(G)$ la cardinalité de la plus grande anticlique de G.

Question 1. Soit G un graphe quelconque. Montrer $\chi(G) \geq \omega(G)$.

Question 2. Soit $G = (V, E)$ un graphe quelconque.

a. Montrer qu'un coloriage valide de G avec c couleurs existe si et seulement si il existe une partition $\{A_1, \ldots, A_c\}$ de V en c anticliques.

b. Montrer $\chi(G)\alpha(G) \geq |V|$.

Graphe parfait. Un graphe G est dit *parfait* si tous ses sous-graphes induits $G[W]$ satisfont : $\chi(G[W]) = \omega(G[W])$.
Graphe imparfait minimal. Un graphe G est dit *imparfait minimal* s'il n'est pas parfait, et que tous ses sous-graphes induits propres sont parfaits.

Question 3. Donner un exemple de graphe parfait, et de graphe imparfait minimal.

Pour simplifier les notations, dans les questions 4 à 8, on fixe G un graphe imparfait minimal quelconque. Sans perte de généralité, on pose $V = \{1, \ldots, n\}$. On note $\alpha = \alpha(G)$, $\omega = \omega(G)$, $\chi = \chi(G)$.

Question 4. Soit A une anticlique de G. Montrer $\omega(G[V \setminus A]) = \omega$.

Question 5. Soit A_0 une anticlique de G de cardinalité α. Montrer qu'il existe $\alpha\omega$ anticliques $A_1, \ldots, A_{\alpha\omega}$, telles que pour chaque sommet $v \in V$, v fait partie d'exactement α anticliques parmi $A_0, \ldots, A_{\alpha\omega}$. (Formellement : $\forall v \in V, |\{i \in \{0, \ldots, \alpha\omega\} : v \in A_i\}| = \alpha$.)

Question 6. On considère une suite d'anticliques $A_0, \ldots, A_{\alpha\omega}$ définie comme dans la question précédente. Montrer que pour tout i dans $\{0, \ldots, \alpha\omega\}$, il existe une clique C_i telle que $C_i \cap A_i = \varnothing$, et $\forall j \neq i, |C_i \cap A_j| = 1$.
Indication : utiliser la question 4

Matrice d'incidence. Étant donné une suite $V_0, \ldots, V_{\alpha\omega}$ de sous-ensembles de $V = \{1, \ldots, n\}$, on définit la *matrice d'incidence* de la suite (V_i) comme la matrice $M = (M_{i,j})_{1 \leq i \leq n, 0 \leq j \leq \alpha\omega} \in \{0,1\}^{n \times (\alpha\omega+1)}$ définie par $M_{i,j} = 1$ si $i \in V_j$, 0 sinon.

Question 7. Soit M_A la matrice d'incidence de la suite $A_0, \ldots, A_{\alpha\omega}$ de la question 5, et M_C la matrice d'incidence de la suite $C_0, \ldots, C_{\alpha\omega}$ de la question 6. On note $M_A^\intercal$ la transposée de M_A.

Montrer que $M_A^\intercal M_C$ est de rang $\alpha\omega + 1$, où les matrices sont vues comme à coefficient dans $\mathbb{Q}$.

Question 8. Montrer $n \geq \alpha\omega + 1$.

Dans les questions suivantes, $G = (V, E)$ est un graphe quelconque.

Question 9. Montrer que G est parfait si et seulement si $\omega(G[W])\alpha(G[W]) \geq |W|$ pour tout sous-graphe induit $G[W]$ de G.

Question 10. Soit $\overline{G} = (V, \mathcal{P}_2(V) \setminus E)$ le graphe *complémentaire* de G. Montrer que G est parfait si et seulement si $\overline{G}$ est parfait.

Inférence de Types

Soit $\mathcal{A} = \{A, B, C, \dots\}$ un ensemble infini de *variables de types*.
On considère $\mathcal{T}$ l'ensemble des *types* définis inductivement par :

- $\mathbb{N} \in \mathcal{T}$. ($\mathbb{N}$ est un type)
- $\mathcal{A} \subseteq \mathcal{T}$. (une variable de types est un type)
- si $T_1 \in \mathcal{T}$ et $T_2 \in \mathcal{T}$ alors $T_1 \to T_2 \in \mathcal{T}$. (la "flèche" de deux types est un type)

Une *substitution de types* $\sigma : \mathcal{A} \longrightarrow \mathcal{T}$ est une fonction qui vaut l'identité presque partout, c'est-à-dire telle que $\{A \in \mathcal{A} \mid \sigma(A) \neq A\}$ est fini.
Si σ est une substitution de types et $T \in \mathcal{T}$ un type, on note $\sigma_\uparrow(T)$ (par abus de notation, on s'autorisera à écrire $\sigma(T)$) le type obtenu inductivement par :

- $\sigma_\uparrow(\mathbb{N}) = \mathbb{N}$.
- pour $\mathcal{A}$, $\sigma_\uparrow(A) = \sigma(A)$.
- pour tout $T_1, T_2 \in \mathcal{T}^2$, $\sigma_\uparrow(T_1 \to T_2) = \sigma_\uparrow(T_1) \to \sigma_\uparrow(T_2)$

Un *système d'équations de types* (ou juste *système*) est un ensemble fini $\{(T_k, T'_k)\}_{1 \leq k \leq n}$ de couples de types.
Un *unificateur* d'un système $\{(T_k, T'_k)\}_{1 \leq k \leq n}$ est une substitution de types σ telle que $\forall k, \sigma_\uparrow(T_k) = \sigma_\uparrow(T'_k)$.

Par exemple, on pose $S_0 = \{(B \to A, C),\ (\mathbb{N}, B)\}$

et σ_0 définie par $\begin{cases} \sigma_0(B) & = & \mathbb{N} \\ \sigma_0(C) & = & \mathbb{N} \to A \\ \sigma_0(X) & = & X \text{ pour tout} X \notin \{B, C\} \end{cases}$

Alors σ_0 est un unificateur de S_0 car :

1. $\sigma_0(B \to A) = \mathbb{N} \to A$ et $\sigma_0(C) = \mathbb{N} \to A$
2. $\sigma_0(\mathbb{N}) = \mathbb{N}$ et $\sigma_0(B) = \mathbb{N}$

Question 1. Trouver si possible un unificateur des systèmes suivants :

1. $\{(\mathbb{N} \to A, B \to \mathbb{N})\}$
2. $\{(\mathbb{N} \to A, B \to (C \to \mathbb{N})),\ (\mathbb{N} \to \mathbb{N}, C)\}$
3. $\{(A \to B), (B \to A)\}$
4. $\{(\mathbb{N} \to (A \to \mathbb{N}), \mathbb{N} \to B),\ (B, A)\}$

Question 2. Un système S admet-il toujours un unificateur ?

Quand un unificateur pour S est-il unique ?

Question 3. En considérant les différentes possibilités pour les formes des types T_1 et T'_1, exprimer l'existence d'un unificateur pour $\{(T_k, T'_k)\}_{1 \leq k \leq n}$ en fonction de l'existence d'un unificateur pour un autre système, bien choisi.

Question 4. Rédiger l'algorithme induit par la question précédente, étudier sa terminaison.

Soit $\mathcal{X} = \{x, y, z, \dots\}$ un ensemble de *variables de termes*.
On considère un langage d'expressions fonctionnelles $\mathcal{E}$ défini inductivement par :

- $\mathbb{N} \subseteq \mathcal{E}$, (les entiers sont des expressions)
- $\mathcal{X} \subseteq \mathcal{E}$ (les variables de termes sont des expressions)
- Si $(E_1, E_2) \in \mathcal{E}^2$, $E_1 + E_2 \in \mathcal{E}$ (la somme de deux expressions est une expression)
- Si $(E_1, E_2) \in \mathcal{E}^2$, $E_1(E_2) \in \mathcal{E}$ (l'application d'une expression à une expression est une expression)
- Si $x \in \mathcal{X}$ et $E \in \mathcal{E}$ alors $(x \mapsto E) \in \mathcal{E}$ (si E est une expression et x une variable, l'expression fonctionnelle $x \mapsto E$ est une expression)

On supposera que les variables x apparaissant dans des sous-expressions $x \mapsto E'$ d'une expression E sont toutes distinctes et distinctes deux à deux des variables de E qui n'apparaissent jamais directement à gauche d'un $\mapsto$. On note $FV(E)$ ce dernier ensemble de variables.

Par exemple, $f_0 = (x \mapsto (y \mapsto x(y) + 1))$ est un élément de $\mathcal{E}$.

Les contextes de types sont des fonctions de $D \subseteq \mathcal{X}$ dans T.
On note $\emptyset$ le contexte de domaine vide et $(\Gamma, x : T)$, le contexte Γ' défini par $\Gamma'(x) = T$ et pour tout y dans le domaine de Γ, $y \neq x \Rightarrow \Gamma'(y) = \Gamma(y)$.

Dans la suite on s'intéresse au problème de donner un type de $\mathcal{T}$ à une expression de $E \in \mathcal{E}$ avec un contexte Γ qui sert d'oracle donnant le type des variables de $FV(E)$, quand c'est possible. On dit qu'on *infère un type de E dans le contexte Γ*.

Question 5. Donner deux types différents qu'il semble légitime de donner à f_0 dans le contexte $\emptyset$.

Question 6. Donner des règles qui formalisent quand il est légitime de donner un type T à E dans le contexte Γ.

Question 7. En déduire un algorithme qui infère un type d'une expression E.

Question 8. Utiliser cet algorithme pour trouver un type de $\mathbf{S} = x \mapsto (\ y \mapsto (\ z \mapsto x(z)(y(z))\ )\ )$ dans le contexte $\emptyset$.

Permutations triables par pile

On s'intéresse aux permutations $\mathfrak{P}(n)$ de $\{1,\dots,n\}$. Une permutation $\pi \in \mathfrak{P}(n)$ est assimilée à la suite $(\pi(1),\pi(2),\dots,\pi(n))$.

Machine à pile. Une *machine à pile* maintient en interne une structure de pile, initialement vide. Elle prend en entrée une permutation $(\pi(1),\dots,\pi(n))$, et effectue une suite fixée d'opérations empile et dépile.

- empile : lit le premier élément non encore lu de la permutation, et l'ajoute à la pile. La i-ième opération empile ajoute donc $\pi(i)$ à la pile.
- dépile : enlève le dernier élément ajouté à la pile, et le renvoie en sortie.

La sortie de la machine est une suite d'éléments de $\{1,\dots,n\}$ renvoyés par les opérations dépile, pris dans l'ordre où ils sont renvoyés. *Exemple :* la machine EEDEDD qui effectue (empile, empile, dépile, empile, dépile, dépile) avec en entrée la permutation $(3,1,2)$ renvoie $(1,2,3)$:

$$\mathsf{EEDEDD}(3,1,2) = (1,2,3).$$

On remarque que le comportement d'une machine à pile est entièrement défini par la suite (fixe) d'opérations empile et dépile qu'elle effectue.

Suite valide. Une telle suite d'opérations est dite *valide* si elle contient exactement n opérations empile et n opérations dépile, et si à tout instant, le nombre d'opérations dépile effectuées est inférieur ou égal au nombre d'opérations empile effectuées. Dans tout le sujet, on ne considère que des machines effectuant des suites valides d'opération.

Permutation triable par pile. Une permutation $\pi \in \mathfrak{P}(n)$ est dite *triable par pile* s'il existe une machine à pile qui prend en entrée $(\pi(1),\dots,\pi(n))$, et qui renvoie $(1,\dots,n)$.

Exemple : l'égalité $\mathsf{EEDEDD}(3,1,2) = (1,2,3)$ plus haut montre que $(3,1,2)$ est triable par pile.

Question 1. Montrer que les permutations $(1,2,3,4)$, $(4,3,2,1)$, $(1,3,2,4)$ sont triables par pile.

Motif. On dit que $\pi \in \mathfrak{P}(n)$ *contient le motif* $\rho \in \mathfrak{P}(k)$ pour $k \leq n$ s'il existe $x_1 < \cdots < x_k$ dans $\{1,\dots,n\}$ tel que $(\pi(x_1),\dots,\pi(x_k)) = (x_{\rho(1)},\dots,x_{\rho(k)})$ (ou plus informellement : tel que $(\pi(x_1),\dots,\pi(x_k))$ et $(\rho(1),\dots,\rho(k))$ sont « dans le même ordre »).

Question 2. Montrer que si π est triable par pile, alors elle ne contient pas le motif $(2,3,1)$.

Question 3. Montrer que si π ne contient pas le motif $(2,3,1)$, alors elle est triable par pile.

Question 4. L'ensemble des permutations triables par pile est-il clos par composition ? Par inverse ?

Question 5. Proposer un algorithme qui détermine en temps linéaire en n si une permutation $\pi \in \mathfrak{P}(n)$ est triable par pile.

Indication : on peut utiliser les questions 2 et 3, mais ce n'est pas obligatoire.

Graphe associé à une permutation. À une permutation $\pi \in \mathfrak{P}(n)$, on associe le graphe non-orienté $G(\pi) = (V, E)$ de sommets $V = \{1, \ldots, n\}$, et d'arêtes $E = \{\{a, b\} \in V : a < b \text{ et } \pi(a) > \pi(b)\}$.
Graphe triable par pile. On dit qu'un graphe $G = (V, E)$ avec $V = \{1, \ldots, n\}$ est *triable par pile* s'il existe $\pi \in \mathfrak{P}(n)$ tel que $G = G(\pi)$ et π est triable par pile.
Graphe réalisable. On dit qu'un graphe $G = (V, E)$ est réalisable s'il est possible de renommer ses sommets V avec les entiers $\{1, \ldots, n\}$, de telle sorte que le graphe obtenu est triable par pile.

Question 6. Montrer que $\pi \mapsto G(\pi)$ est une injection de $\mathfrak{P}(n)$ vers l'ensemble des graphes de sommets $\{1, \ldots, n\}$. Est-ce une bijection ?

Question 7. Donner un exemple de graphe qui n'est pas réalisable.

Question 8. On considère l'ensemble des graphes formés en partant du graphe vide $(\varnothing, \varnothing)$, et en utilisant uniquement les deux opérations suivantes : ajout d'un sommet universel à un graphe de l'ensemble (un sommet est dit *universel* s'il est relié à tous les autres sommets), union disjointe de deux graphes de l'ensemble. Montrer que l'ensemble obtenu est exactement l'ensemble des graphes réalisables.

Question 9. En s'inspirant de la question précédente, proposer un algorithme qui détermine si un graphe $G = (V, E)$ est réalisable, en temps $O(|V|^3)$.

Composition Monadique

On se place dans un langage récursif, purement fonctionnel, avec des définitions de types inductifs et un système de types polymorphes similaires à ceux d'OCaml. On pourra utiliser indifféremment du pseudocode fonctionnel ou de la syntaxe OCaml.
On suppose l'existence :

- d'une fonction identité `id` polymorphe de type $A \to A$;
- d'un opérateur $(\circ) : (A \to B) \to (B \to C) \to (A \to C)$ de composition de fonctions, noté $\circ$ dans sa version infixe ;
- d'un opérateur $\mapsto$ de définition de fonction anonyme (`fun` en OCaml) ;
- un opérateur de reconnaissance de motifs (`match...with` en OCaml) ; et
- de types de bases usuels, comme `int` le type des entiers, `string` le type des chaînes de caractères, et `unit` le type de l'unique élément ().

Un *constructeur de type* est une entité informatique qui prend un type et renvoie un type.
On définit par exemple le constructeur de type `liste` en définissant le type polymorphe `liste` A (où A est une variable de type). A titre d'exemple, on présente une définition de la fonction longueur de type `liste` $A \to$ `int` en pseudocode :

`liste` $A =$ `Nil` | `Cons` A (`liste` A)
`longueur` : `liste` $A \to$ `int`
  `longueur` `Nil` $= 0$
  `longueur` (`Cons` _ qu) $= 1 +$ (`longueur` qu)

---

et en OCaml :

```
type 'a liste = Nil | Cons of ('a * 'a liste)
let rec longueur (l : 'a liste) : int = match l with
        Nil -> 0
      | Cons (_,qu) -> 1 + (longueur qu)
```

Question 1. Définir un constructeur de type `option` tel qu'un élément de `option` A est soit vide, soit un élément de type `A`.

Un constructeur de types `F` est un *foncteur* quand il existe une fonction polymorphe
`fmap` : $(A \to B) \to (F\ A) \to (F\ B)$
qui vérifie les lois suivantes (pour tout f et g) :
`fmap id` $=$ `id`
`fmap` $(f \circ g) =$ (`fmap` f) $\circ$ (`fmap` g)

Attention : Dans ce sujet, on ne demande pas de preuves que les lois sont respectées.

Question 2. Montrer que `option` et `liste` sont des foncteurs.

Un foncteur `M` est une *monade* quand il existe deux fonctions polymorphes
`return` : $A \to (M\ A)$
`bind` : $(M\ A) \to (A \to (M\ B)) \to (M\ B)$
qui vérifient les lois suivantes, pour tout f, g et x :
`bind` (`return` x) $f = f\ x$
`bind` x `return` $= x$
`bind` (`bind` $x\ f$) $g =$ `bind` x $(y \mapsto$ (`bind` $(f\ y)\ g$))

Question 3. Montrer que `option` et `liste` sont des monades.

Pour tout type S, on définit (`etat` S) A comme étant le type polymorphe $S \to (S, A)$
On suppose l'existence d'un type abstrait `memoire` représentant une table d'association entre `string` et `int` avec les deux fonctions suivantes :
`trouve` : `string` $\to$ `memoire` $\to$ (`option int`)
`majour` : `string` $\to$ `int` $\to$ `memoire` $\to$ `memoire`
La première recherche la valeur associée à une clef, la seconde met-à-jour un couple clef-valeur.

Question 4. Montrer que `etat` S est une monade pour tout S.

En déduire l'écriture avec `bind` d'une *procédure* qui prend en entrée trois clefs, récupère successivement deux valeurs d'une mémoire à partir des deux premières clefs puis associe dans la mémoire la somme des deux valeurs récupérée à la troisième clef.

On donne une définition alternative des monades :
Un foncteur `M` est une *monade* quand il existe deux fonctions polymorphes
`return` : $A \to (M\ A)$
`join` : $M\ (M\ A) \to (M\ A)$
qui vérifient les lois suivantes, pour tout f, g et x :
`join` $\circ$ `return` $=$ `id`
`join` $\circ$ (`fmap return`) $=$ `id`
`join` $\circ$ `join` $=$ `join` $\circ$ (`fmap join`)

Question 5. Montrer que les deux définitions sont équivalentes.

Si M_1 et M_2 sont deux constructeurs de types, on définit (`compose` $M_1\ M_2$) $A = M_1\ (M_2\ A)$.

Question 6. Donner une condition suffisante pour que (`compose` $M_1\ M_2$) soit un foncteur.

Question 7. Donner une condition suffisante pour que (`compose` $M_1\ M_2$) soit une monade.

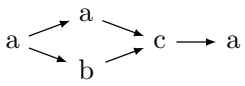
En déduire une nouvelle écriture, plus simple, de la procédure de la question 4.

Mots partiels et Théorème de Dilworth

Mots partiels

Soit Σ un ensemble fini de lettres, et Σ^* l'ensemble des mots de Σ, qui sont des séquences finies de lettres. Nous généralisons la notion de mots à des structures partiellement ordonnées.
Soit $G = (E, R)$ un graphe dirigé fini : E est un ensemble fini et R un sous ensemble de $E \times E$. Un chemin de G est une séquence $(x_1, y_1), \ldots, (x_n, y_n)$ d'éléments de R telle que $y_i = x_{i+1}$. Le graphe G est dit *acyclique* s'il n'existe pas de chemin tel que $y_n = x_1$.
Un mot partiel sur Σ est un triplet (E, R, μ) avec (E, R) un graphe fini acyclique et $\mu \colon E \to \Sigma$. La taille d'un mot partiel est le nombre d'éléments de E.
Soit $p = (E, R, \mu)$ un mot partiel de taille n. Un mot $u_1 \cdots u_n \in \Sigma^*$ généralise p s'il existe $\psi \colon E \to \{0, \ldots, n-1\}$, injective, tel que pour tout $(x, y) \in R$, $\psi(x) < \psi(y)$ et si pour tout $e \in E$, on a $u_{\psi(e)} = \mu(e)$. Dans la suite on note Total(p) l'ensemble des mots qui généralisent p.

Question 1. Que vaut Total(p) pour p dessiné ci-dessous, où chaque sommet est résumé par son image par μ.



Question 2. Donnez une borne supérieure sur la taille de Total(p) en fonction de la taille de Σ.

Soit L un langage de Σ^*. Un mot partiel p sur Σ *appartient à* Partiel(L) s'il existe $v \in$ Total(p) qui appartient à L.

Question 3. Montrez que si L est dans PTIME, alors Partiel(L) appartient à NP.

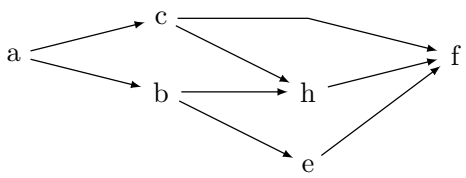
(Unary) 3-partition est un problème qui prend en entrée une suite finie de $3n$ entiers écrits **en unaire** $x_1, \ldots, x_{3n}$ et vérifie s'il existe une partition en n triplets dont les sommes deux à deux sont égales. On admet dans la suite que ce problème est NP-complet.

Question 4. Montrez par une réduction à 3-partition qu'il existe un langage L PTIME tel que Partiel(L) est NP-complet.

Question 5. Montrez que Partiel($\Sigma^* a b^* a \Sigma^*$) est vérifiable en temps polynomial.

Soit $p = (E, R, \mu)$ un mot partiel sur Σ. Une décomposition en chemins de p est un ensemble X de chemins tel que tout sommet de E appartient à exactement un chemin.

Question 6. Donnez une décomposition en chemins avec le moins de chemins possibles pour le mot partiel suivant :



Question 7. Soit L un langage régulier (reconnu par un automate déterministe). Montrez que Partiel(L) est vérifiable en temps $O(n^w)$ avec n la taille du mot partiel et w la taille de sa plus petite décomposition en chemins.

Soit $p = (E, R, \mu)$ un mot partiel sur Σ. Une anti-chaîne de p est un ensemble X tel qu'il n'existe pas de chemin entre deux éléments de X. On appelle largeur de p la taille de la plus grosse anti-chaîne de p.
On admet le théorème suivant :

Théorème 1 (Dilworth). Soit p un mot partiel. La largeur de p est égale à la taille de sa plus petite décomposition en chemins. Cette dernière est calculable en temps polynomial.

Question 8. En déduire que pour les mots partiels de largeur fixée, le problème Partiel(L) pour L un langage régulier est calculable en temps polynomial. Peut-on en déduire que le problème est calculable en temps polynomial, sans la contrainte que la largeur est fixée ?

Terminaison de $\lambda_{\texttt{ref}}$

Le λ-calcul, langage formel modélisant la programmation fonctionnelle, est défini par la syntaxe :
$M, N ::= x \mid M\ N \mid \lambda x.M \mid ()$
où x est issu d'un ensemble infini de *variables de termes*.
Ainsi, un terme M est :

- soit une *variable* x,
- soit une *abstraction* $\lambda x.M$ (qu'il faut comprendre comme la fonction qui au paramètre x associe le terme M), on dit dans ce cas que la variable x est *liée* dans M,
- soit une *application* $M\ N$ (qu'il faut comprendre comme le terme - fonction - M appliqué au terme - argument - N)
- soit l'*unité* (), un terme de base.

Ces termes correspondent respectivement aux expressions OCaml suivantes : `x` un nom, `fun x-> e` une fonction anonyme, `e1 e2` une application et (), l'unité.
On note $M\{N/x\}$ le terme obtenu en remplaçant toutes les occurrences (non-liées) de la variable x dans M par le terme N.
Une relation d'*α-conversion* $\equiv_\alpha$ autorise le renommage des variables liées : si $y \notin M$, alors $\lambda x.M \equiv_\alpha \lambda y.(M\{x/y\})$. Dans la suite, on considèrera les termes *modulo α-conversion* (on s'autorisera à renommer les variables liées dans les sous-termes), et on utilisera cette opération pour présenter des termes dans lesquels les variables liées sont distinctes deux à deux, et distinctes des variables non-liées.
Les *contextes d'évaluation* sont définis par : $\mathbf{E} ::= [] \mid M\ \mathbf{E} \mid \mathbf{E}\ M$
Si M est un terme et $\mathbf{E}$ un contexte, on note $\mathbf{E}[M]$ le terme obtenu en remplaçant [] dans $\mathbf{E}$ par M.
La sémantique (le comportement d'un terme) est donnée par une relation de réduction $\longrightarrow$ donnée par :

1. $(\lambda x.M)\ N \longrightarrow M\{N/x\}$ (on applique la fonction qui à x associe M à N, et on récupère le corps de la fonction M dans lequel l'argument N remplace le paramètre x.)
2. si $M \longrightarrow M'$ alors $\mathbf{E}[M] \longrightarrow \mathbf{E}[M']$ (on peut réduire dans un contexte).

Comme en OCaml, l'application est parenthésée à gauche, ainsi $M_1\ M_2\ M_3$ désigne $(M_1\ M_2)\ M_3$ et on écrit $\lambda xy.M$ pour $\lambda x.(\lambda y.M)$
$\longrightarrow^*$ désigne la clôture réflexive et transitive de $\longrightarrow$. Les *réduits* d'un terme M sont les éléments de l'ensemble $\{M' \mid M \longrightarrow^* M'\}$.

Question 1. Soit $\delta = \lambda x.(x\ x)$ et $I = \lambda x.x$. Expliciter les réduits du terme $A = (\lambda x.I)\ (\delta\ \delta)$.

On donne un système de *types simples* au λ-calcul. Les types sont donnés par :
$S, T ::= S \to T \mid \texttt{unit}$
Ainsi un type est soit le type fonctionnel $S \to T$ des fonctions de S dans T soit `unit` le type de base.
Une *hypothèse* $x : T$ associe une variable de terme à un type. Un *contexte de typage* Γ est un ensemble d'hypothèses et on note $\Gamma, x : T$ le contexte $\Gamma \cup \{x : T\}$ quand x n'est pas dans Γ. Un *jugement* de typage $\Gamma \vdash M : T$ indique que le terme M a le type T dans le contexte Γ (on dit qu'un terme est *typable* s'il existe Γ, T tel que $\Gamma \vdash M : T$).

Les règles de typage permettant de déduire un jugement sont données par :

1. $\Gamma \vdash () : \texttt{unit}$
2. $\Gamma, x : T \vdash x : T$
3. si $\Gamma, x : T_1 \vdash M : T_2$ alors $\Gamma \vdash \lambda x.M : T_1 \to T_2$
4. si $\Gamma \vdash M : T_1 \to T_2$ et $\Gamma \vdash N : T_1$, alors $\Gamma \vdash M\ N : T_2$

On note $\lambda_{\texttt{ST}}$ l'ensemble des termes typables.

Question 2. Montrer que si M est typable et $M = \mathbf{E}[N]$, alors N est typable, puis expliquer pourquoi A n'est pas typable.

On dispose d'un ensemble infini *d'adresses* $\mathcal{A}$. Une mémoire σ est une fonction qui a chaque élément d'un sous-ensemble fini de $\mathcal{A}$ (appelé son *support*) associe un λ-terme.
On définit un λ-calcul appelé $\lambda_{\texttt{ref}}$ contenant une valeur () de type `unit` et trois opérateurs qui permettent de manipuler une mémoire, inspirés de leurs homologues en OCaml :

1. une opération `ref` de référencement qui prend un λ-terme, le stocke en mémoire à une nouvelle adresse α et renvoie α.
2. une opération `deref` (! en OCaml) de déréférencement qui prend une adresse et renvoie le terme contenu à cette adresse en mémoire.
3. une opération `assig` (:= en OCaml) d'assignation qui prend une adresse α et un terme M, modifie la mémoire pour remplacer la valeur en α par M, et renvoie `unit`.

Question 3. Donner des règles de typage pour les termes de $\lambda_{\texttt{ref}}$ et les mémoires.

Question 4. Donner une sémantique pour $\lambda_{\texttt{ref}}$ explicitant comment réduire un couple composé d'un terme typable et d'une mémoire.

Les réductions qui consomment un opérateur `ref`, `deref` ou `assig` sont appelée *impures*, les autres *pures*.

Question 5. Définir une fonction *d'élagage* $\mathbf{p}$ qui associe à chaque terme typable de $\lambda_{\texttt{ref}}$ un terme typable de $\lambda_{\texttt{ST}}$ telle que si $(M, \sigma) \longrightarrow (M', \sigma)$ avec une réduction pure, alors $\mathbf{p}(M) \longrightarrow \mathbf{p}(M')$.

Un terme M est *terminant* quand il n'existe pas de suite infinie $(M_n)_{n\in\mathbb{N}}$ telle que
$M_0 = M$ et $\forall n \in \mathbb{N}, M_n \longrightarrow M_{n+1}$.
On admettra le résultat suivant :
Si M est un terme de $\lambda_{\texttt{ST}}$, alors M est terminant.

Question 6. Montrer que toute chaîne de réduction infinie dans $\lambda_{\texttt{ref}}$ contient une infinité de réductions impures. Proposer un terme de $\lambda_{\texttt{ref}}$ typable et non-terminant.

On divise la mémoire en *régions* identifiée par des entiers naturels. Les opérateurs $\texttt{ref}_n$, $\texttt{deref}_n$ et $\texttt{assign}_n$ sont maintenant indexés par la région qu'ils manipulent.

Question 7. Modifier le système de types pour qu'il associe à un terme typable son *effet* c'est à dire la région la plus haute qu'une réduction de ce terme peut manipuler (en effectuant une réduction d'un des trois opérateurs impurs).

Question 8. En contraignant les types par les régions, délimiter un sous-ensemble de $\lambda_{\texttt{ref}}$ terminant.

Graphes d'influence

On note $[\![a, b]\!]$ l'intervalle entier $\{a, a+1, \ldots, b\}$. On note $|S|$ la cardinalité d'un ensemble S.
Graphe simple. Dans tout l'énoncé, on considère des graphes simples : c'est-à-dire qu'ils sont définis par un ensemble de sommets V, et un ensemble d'arêtes E, et chaque arête est une paire non ordonnée $\{i, j\}$ d'éléments distincts de V.
Isomorphisme de graphe. Deux graphes $G = (V, E)$ et $G' = (V', E')$ sont dits isomorphes s'il existe une bijection $\varphi : V \to V'$ telle que $E' = \{\{\varphi(i), \varphi(j)\} : \{i, j\} \in E\}$.
Graphe d'influence. Soit $\vec{v} = (v_1, \ldots, v_n) \in \mathbb{R}^n$. Au vecteur $\vec{v}$, on associe le graphe $\mathsf{Infl}(\vec{v}) = (V, E)$ avec pour sommets $V = [\![1, n]\!]$, et pour arêtes $E = \{\{i, j\} : |v_i - v_j| < 1\}$. Un graphe $G = (V, E)$ est un *graphe d'influence* s'il existe $\vec{v} \in \mathbb{R}^{|V|}$ tel que G est isomorphe à $\mathsf{Infl}(v)$.

Informellement, on peut penser aux sommets i d'un graphe d'influence comme étant des *agents* ; v_i est l'*opinion* du i-ième agent ; et un agent *influence* un autre agent si leurs opinions sont suffisamment proches.

Question 1. Donner un exemple simple de famille infinie de graphes d'influence (deux à deux non isomorphes). Donner une exemple de graphe qui n'est pas un graphe d'influence.

Question 2.

1. Montrer que tout graphe d'influence $G = (V, E)$ est un *graphe d'incomparabilité* : c'est-à-dire qu'il existe un ordre partiel $\preceq$ sur V tel que $\{i, j\} \in E$ si et seulement si $i \not\preceq j$ et $j \not\preceq i$.
2. Montrer que tout graphe d'influence $G = (V, E)$ est *cordal* : c'est-à-dire que pour toute suite de sommets $k_1, \ldots, k_m$ formant un cycle de longueur $m > 3$, il existe $i, j \in [\![1, m]\!]$ tels que $|j - i| \neq 1 \mod m$ et $\{k_i, k_j\} \in E$.

Pour $G = (V, E)$ un graphe et $i \in V$, on note $G[i] = \{j \in V : \{i, j\} \in E\} \cup \{i\}$ le *voisinage* de i.
Soit :

$$f : \mathbb{R}^n \to \mathbb{R}^n$$
$$(v_i)_{i \in [\![1,n]\!]} \mapsto \left(\frac{\sum_{j \in \mathsf{Infl}(\vec{v})[i]} v_j}{|\mathsf{Infl}(\vec{v})[i]|} \right)_{i \in [\![1,n]\!]}.$$

Autrement dit, f remplace l'opinion de chaque agent par la moyenne des opinions des agents qui l'influencent (lui-même compris).
Chaîne d'influence. Une *chaîne d'influence* est une suite de vecteurs $\vec{v}^k \in \mathbb{R}^n$ telle que pour tout $i \in \mathbb{N}$, $\vec{v}^{k+1} = f(\vec{v}^k)$.
Dans toute la suite, on supposera que le vecteur initial $\vec{v}^0$ d'une chaîne d'influence est ordonné par ordre croissant : $v_1^0 \leq v_2^0 \leq \cdots \leq v_n^0$.

Question 3. Soit $(\vec{v}^k)_{k \in \mathbb{N}}$ une chaîne d'influence.

1. Montrer que pour tout $k \in \mathbb{N}$, $v_1^k \leq v_2^k \leq \cdots \leq v_n^k$.
2. Montrer que $(v_1^k)_{k \in \mathbb{N}}$ est croissante, et $(v_n^k)_{k \in \mathbb{N}}$ est décroissante.

Question 4. Soit $(\vec{v}^k)_{k \in \mathbb{N}}$ une chaîne d'influence. Pour $i \in [\![1, n]\!]$ fixé, on définit la suite u_k par $u_k = 1$ si l'arête $\{i, i+1\}$ existe dans $\mathsf{Infl}(\vec{v}^k)$, et $u_k = 0$ sinon. Montrer que $(u_k)_{k \in \mathbb{N}}$ est stationnaire.

Rappel. Une suite $(a_k)_{k \in \mathbb{N}}$ est dite *stationnaire* s'il existe $b \in \mathbb{N}$ tel que $\forall k \geq b, a_k = a_b$.

Question 5. Montrer que pour toute chaîne d'influence, après un nombre fini d'itérations de f, plus aucun agent ne change d'opinion. (Autrement dit, toute chaîne d'influence est stationnaire.)

Question 6. Donner un algorithme qui, étant donné en entrée une suite stationnaire de graphes $(G_i)_{i\in\mathbb{N}}$ (représentée par un segment initial au-delà duquel la suite reste constante) :

— termine et renvoie `VRAI` dans le cas où il existe $\vec{v}^0 \in \mathbb{R}^n$ tel que $(G_i)_{i\in\mathbb{N}}$ est la suite des graphes d'influence de la chaîne d'influence initiée par $\vec{v}^0$;

— renvoie `FAUX` ou ne termine pas dans le cas contraire.

Question 7. Montrer que pour tout $n \in \mathbb{N}$, il existe un nombre fini de suites de graphes qui sont les graphes d'influence successifs d'une chaîne d'influence dans $\mathbb{R}^n$.

Raffinement d'évènements

On se place dans un langage récursif, purement fonctionnel, avec des définitions de types inductifs et un système de types polymorphes similaires à ceux d'OCaml. On pourra utiliser indifféremment du pseudocode fonctionnel ou de la syntaxe OCaml.
On suppose l'existence :

- de types de bases usuels, comme `Bool` le type des booléens, `Int` le type des entiers, `String` le type des chaînes de caractères, et `Unit` le type de l'unique élément ().
- de définitions de types paramétrés avec des constructeurs, comme :
  `type ArbreB A = Feuille | Noeud A (ArbreB A) (ArbreB A)`
- de types enregistrements $\{\mathtt{a_1} : \mathtt{T_1}, \ldots, \mathtt{a_n} : \mathtt{T_n}\}$ qui permettent de définir, par exemple, les points du plan avec `type Point = P {abscisse : Int, ordonnee : Int}` qu'on peut utiliser avec, par exemple, `origine = P {abscisse = 0, ordonnee = 0}`.
- de types listes natifs [A] avec les fonctions de manipulation usuelles (construction, longueur, tête, queue, ...).

Un *type de machine* est un type abstrait, qui représente l'état d'un système. On utilisera `M` pour représenter un tel type et `m` pour représenter des éléments de ce type, appelés *machines*. L'*invariant* d'une machine est un prédicat sur un type de machine, de type $\mathtt{M} \to \mathtt{Bool}$ qui décrit les propriétés qu'on attend d'un élément de `M` ; on supposera qu'il est unique (pour chaque type `M`, on définit un unique invariant `inv_M`).
Un *évènement* d'un type de machine `M` d'entrée `A` et de sortie `B` est la donnée d'un prédicat sur les machines et les entrées, appelé *garde*, et d'une fonction des entrées et des machines dans les sorties et les machines :

```
type Evenement M A B = E {
                       garde : A → M → Bool
                       action : A → M → (B, M)
                       }
```

L'idée est qu'étant donnée une machine, un évènement et une entrée, si la garde de l'évènement est vraie pour la machine et l'entrée, on peut appliquer l'action de l'évènement à la machine et l'entrée et récupérer une nouvelle machine (la machine dans un nouvel état) et une sortie.
Un évènement est *sûr* quand l'implication suivante est vraie : "Si l'invariant de la machine est vrai, et la garde est vraie pour la machine et l'entrée, alors l'invariant de la nouvelle machine obtenue après l'action est vrai".

Question 1. On considère comme premier type de machine, les *compteurs bornés*. Un compteur borné est composé de deux entiers : une *valeur maximale*, fixe et positive, et une *valeur actuelle*, comprise entre 0 et la valeur maximale.

Donner une définition des compteurs bornés, de leur invariant, et d'évènements d'incrémentation et de décrémentation. Prouver leur sûreté.

Un *raffinement* $\mathcal{R}$ entre les types de machine $\mathtt{M_1}$ et $\mathtt{M_2}$ est une relation entre les éléments de $\mathtt{M_1}$ et $\mathtt{M_2}$ telle que si $(\mathtt{m_1}, \mathtt{m_2}) \in \mathcal{R}$ et l'invariant de $\mathtt{m_2}$ est vrai, alors l'invariant de $\mathtt{m_1}$ est vrai.

Question 2. On dispose d'une fonction `lift` : $\mathtt{M}_2 \to \mathtt{M}_1$. Quelle condition sur `lift` permet de construire un raffinement (non-trivial) entre $\mathtt{M}_1$ et $\mathtt{M}_2$?

L'inverse est-il vrai ?

Question 3. Donner un type de machine représentant les piles de taille bornée, puis exhiber un raffinement entre les compteurs bornés et ces piles bornées.

Soit un raffinement $\mathcal{R}$ entre $\mathtt{M}_1$ et $\mathtt{M}_2$.
Soit deux évènements $\mathtt{e}_1$: `Evenement` $\mathtt{M}_1$ $\mathtt{A}_1$ $\mathtt{B}_1$ et $\mathtt{e}_2$: `Evenement` $\mathtt{M}_2$ $\mathtt{A}_2$ $\mathtt{B}_2$.
On dit que $\mathtt{e}_2$ *raffine* $\mathtt{e}_1$ *vis-à-vis* de $\mathcal{R}$ (ou simplement e_2 *raffine* e_1 quand il est clair de quel raffinement $\mathcal{R}$ on parle) quand il existe deux fonctions
$\mathtt{lift\_in} : \mathtt{A}_2 \to \mathtt{A}_1$
$\mathtt{lift\_out} : \mathtt{B}_2 \to \mathtt{B}_1$
qui vérifient :

- *Renforcement* : Pour tout $\mathtt{x}, \mathtt{m}_1, \mathtt{m}_2$ tels que $\mathtt{m}_1 \ \mathcal{R} \ \mathtt{m}_2$, si $\mathtt{e}_2.\mathtt{garde}\ \mathtt{x}\ \mathtt{m}_2$, alors $\mathtt{e}_1.\mathtt{garde}\ (\mathtt{lift\_in}\ \mathtt{x})\ \mathtt{m}_1$.
- *Simulation* : Pour tout $\mathtt{x}, \mathtt{m}_1, \mathtt{m}_2$ tels que $\mathtt{m}_1 \ \mathcal{R} \ \mathtt{m}_2$ et $\mathtt{e}_2.\mathtt{garde}\ \mathtt{x}\ \mathtt{m}_2$,
  si $(\mathtt{b}_2, \mathtt{m}'_2) = \mathtt{e}_2.\mathtt{action}\ \mathtt{x}\ \mathtt{m}_2$
  et $(\mathtt{b}_1, \mathtt{m}'_1) = \mathtt{e}_1.\mathtt{action}\ \mathtt{x}\ \mathtt{m}_1$,
  alors $\mathtt{m}'_1 \ \mathcal{R} \ \mathtt{m}'_2$ et $\mathtt{b}_1 = \mathtt{lift\_out}\ \mathtt{b}_2$.

Question 4. Montrer que si $\mathtt{e}_2$ raffine $\mathtt{e}_1$ et $\mathtt{e}_1$ est sûr, alors $\mathtt{e}_2$ est sûr.

Question 5. Donner deux évènements pour les piles bornées qui raffinent les évènements des compteurs bornés vis-à-vis du raffinement de la Question 3.

On définit le type $\mathtt{Mealy\ A\ B} = \mathtt{M}\ \{\mathtt{run} : \mathtt{A} \to (\mathtt{B}, \mathtt{Mealy\ A\ B})\}$.

Question 6. Le type `Mealy A B` est-il un type inductif ?

Question 7. Exprimer la définition de l'empilage dans une pile bornée en utilisant le type `Mealy A B` (pour un `A` et un `B` bien choisis).

Question 8. Tenter d'en induire une fonction `evt_vers_Mealy` qui prend un évènement et le transforme en *machine de Mealy* (en une entité qui utilise un type `Mealy A B`).

Réseaux booléens et leurs points fixes

Un *réseau booléen* à n composantes est une fonction $f : \{0,1\}^n \to \{0,1\}^n$. Une *configuration* de f est un n-uplet $x \in \{0,1\}^n$. La i-ème composante de x est notée x_i. Pour $1 \leq i \leq n$, la i-ème composante de $f(x)$ est notée $f_i(x)$, où f_i est une fonction de $\{0,1\}^n$ dans $\{0,1\}$. On note également e_j le n-uplet ne comportant qu'un seul "1" en position j.
L'ensemble des configurations est muni de la relation d'ordre partielle : $x \leq y \iff \forall i, x_i \leq y_i$. Si x et y sont deux configurations, on note $x + y$ l'addition modulo 2 composante à composante.

Un *graphe signé* est un couple $G = (V, E)$ où $E \subseteq V \times V \times \{-1, 1\}$. V est l'ensemble des sommets, E l'ensemble des arcs, qui peuvent être positifs (signe 1) ou négatifs (signe -1). Il est possible d'avoir à la fois un arc positif et un arc négatif d'un sommet vers un autre. On note $|G|$ le graphe orienté obtenu en ignorant les signes. On étend la notion de signe à tout chemin de G en faisant le produit des signes des arcs.

Le *graphe d'interaction* d'un réseau booléen f est le graphe signé $G(f)$ défini comme suit. L'ensemble des sommets est $V = \{1, \ldots, n\}$. Pour tout $i, j \in V$, il existe un arc positif (resp. négatif) $j \to i$ si et seulement si il existe une configuration x telle que $x_j = 0$ et $f_i(x + e_j) - f_i(x)$ est strictement positif (resp. négatif). En d'autres termes, un arc $j \to i$ signifie que f_i dépend de la j-ème variable. Si l'arc est positif, on dit que j est un *activateur* de i. Si l'arc est négatif, on dit que j est un *inhibiteur* de i.

Question 1. Soit le réseau booléen à 3 composantes suivant :

$$f_1(x) := x_3 \wedge (\neg x_1 \vee \neg x_2)$$
$$f_2(x) := x_3 \wedge x_1$$
$$f_3(x) := x_2$$

Calculer la sortie de f sur chaque entrée possible et donner le graphe $G(f)$.

On note : $x \leq_i^G y$ si et seulement si $x_j \leq y_j$ pour tous les activateurs j de i, et $x_j \geq y_j$ pour tous les inhibiteurs j de i.
Pour deux configurations x, y, on note $\Delta(x, y) \subseteq \{1, \ldots, n\}$ l'ensemble des positions où elles diffèrent, et la *distance de Hamming* entre x et y est $|\Delta(x, y)|$.

Question 2. Soit f un réseau Booléen et G son graphe d'interactions. Soit $x \neq y$ tels que $x \leq_i^G y$.

1. Supposons qu'il existe j tel que $x_j < y_j$. Montrer que $x + e_j \leq_i^G y$ et $f_i(x + e_j) \geq f_i(x)$.
2. Supposons qu'il existe j tel que $y_j < x_j$. Montrer que $x \leq_i^G y + e_j$ et $f_i(y + e_j) \leq f_i(y)$.

Dans les questions suivantes on considère un graphe signé à n sommets $\{1, \ldots, n\}$ noté G, et on note $F(G)$ l'ensemble des réseaux booléens qui ont G pour graphe d'interactions.

Question 3. Montrer que si $f \in F(G)$, alors pour tout $1 \leq i \leq n$ et toutes configurations x, y :

$$x \leq_i^G y \implies f_i(x) \leq f_i(y)$$

On procédera par induction sur la distance de Hamming entre x et y.

Pour un réseau booléen f, un *point fixe* de f est une configuration x telle que $f(x) = x$.
De plus, pour un graphe G on note $G[U]$ le sous-graphe induit par le sous-ensemble de sommets U, c'est-à-dire en retirant les sommets hors de U et les arcs attachés.

Question 4. Soit $f \in F(G)$ avec deux points fixes distincts x et y. Montrer qu'il existe deux sommets i, j (pas nécessairement distincts) tels que G a un arc $j \to i$ de signe $(y_j - x_j)(y_i - x_i)$.

Question 5. Montrer que si $f \in F(G)$ a deux points fixes distincts x et y, alors $G[\Delta(x, y)]$ possède un cycle positif.

On admet le théorème suivant.

> Si G est fortement connexe et sans cycle négatif, alors il existe une partition de ses sommets en deux ensembles A et B telle qu'un arc (u, v) de G est positif si et seulement si u et v sont dans la même partie.

Un réseau est dit *monotone* si pour toutes configurations x, y :

$$x \leq y \implies f(x) \leq f(y)$$

Question 6. Montrer que si G est fortement connexe et sans cycle négatif, alors il existe une configuration z telle que, pour tout $f \in F(G)$, le réseau $h(x) := f(x + z) + z$ est monotone.

Montrer que le graphe d'interactions de h s'obtient à partir de celui de G en rendant positifs tous les arcs.

Indice : Commencer avec une partition des sommets A, B donnée par le théorème. Définir z par $z_i = 1$ si $i \in A$ et $z_i = 0$ sinon.

Question 7. Montrer que si G est fortement connexe et sans cycle négatif, alors tout $f \in F(G)$ a au moins deux points fixes.

Question 8. Montrer que si G n'a pas de cycle négatif, alors tout $f \in F(G)$ a au moins un point fixe.

Résolution efficace de systèmes linéaires creux

Notation. Soit $\mathbb{K} = \mathbb{Z}/2\mathbb{Z}$ le corps à deux éléments. On note $\deg(P)$ le degré d'un polynôme $P \in \mathbb{K}[X]$. Étant donné $P, Q \in \mathbb{K}[X]$, on note P rmod Q le reste dans la division euclidienne de P par Q. Un vecteur $x \in \mathbb{K}^n$ est implicitement vu commes un vecteur colonne ; sa transposée x^t est un vecteur ligne. La cardinalité d'un ensemble E est notée $|E|$.
Poids de Hamming. Le *poids de Hamming* d'un vecteur $v \in \mathbb{K}^n$, noté $\mathsf{H}(v)$, est le nombre de coordonnées non-nulles du vecteur. De même, le poids de Hamming d'une matrice est le nombre d'entrées non-nulles de la matrice.
Problème de résolution linéaire. Le *problème de résolution linéaire* prend en entrée une matrice $M \in \mathbb{K}^{n\times n}$, et un vecteur $b \in \mathbb{K}^n$. Il demande de trouver un vecteur $x \in \mathbb{K}^n$ tel que $Mx = b$, si un tel x existe, ou de renvoyer $\perp$ sinon.
Représentation par indice. Soit $M = (m_{i,j}) \in \mathbb{K}^{m\times n}$ une matrice. La *représentation par indice* de M est l'ensemble $\{(i, j) : m_{i,j} = 1\}$. Noter que pour m et n fixés, cette représentation définit la matrice de manière unique, en utilisant $O(\mathsf{H}(M))$ entiers.
Dans tout l'énoncé, on supposera que la matrice M en entrée du problème de résolution linéaire est donnée sous forme de sa représentation par indice.

Question 1. Esquisser sans détailler un algorithme en temps $O(n^3)$ pour résoudre le problème de résolution linéaire.

Question 2. Soit $G = (V, E)$ un graphe formé d'un ensemble de sommets V, et un ensemble d'arêtes E. Les arêtes de E sont des paires non-ordonnées $\{v, w\}$ d'éléments de V, avec $v \neq w$. (On suppose que E est implémenté sous forme d'une liste de paires.) Proposer un algorithme en temps $O(|V| + |E|)$ qui calcule la partition de V en composantes connexes.

Indication : on pourra d'abord créer un dictionnaire qui à chaque sommet associe la liste de ses voisins.

Question 3.

a. Soit $M \in \mathbb{K}^{n\times n}$ une matrice telle que chaque *ligne* de M a un poids de Hamming au plus 2. Proposer un algorithme qui résout le problème de résolution linéaire pour une telle matrice M, en temps $O(n)$.

b. Même question si chaque *colonne* de M, et non plus chaque ligne, a un poids de Hamming au plus 2.

Indication : Dans les deux cas, il peut être utile de définir un graphe bien choisi (pas nécessairement le même), et de chercher d'abord un algorithme en temps $O(n^2)$, avant de raffiner en temps linéaire.

Polynôme minimal d'une matrice. Étant donné une matrice $M \in \mathbb{K}^{n\times n}$, son *polynôme minimal* est le polynôme $P = \sum_{i=0}^{d} \lambda_i X^i \in \mathbb{K}[X]$ de degré $d \leq n$ minimal tel que $P(M) = \sum_{i=0}^{d} \lambda_i M^i = 0$.

Question 4. Proposer un algorithme qui résout le problème de résolution linéaire en temps $O(n^2 + n\mathsf{H}(M))$, lorsque la matrice M en entrée du problème est inversible, et que son polynôme minimal est connu.

Indication : trouver une expression de M^{-1} sous forme d'un polynôme en M.

Matrice de Hankel. Soit $(a_0, \ldots, a_{2n-1})$ une suite d'éléments de $\mathbb{K}$. La *matrice de Hankel* générée par cette suite est la matrice $M = (m_{i,j}) \in \mathbb{K}^{n\times(n+1)}$, avec n lignes et $n+1$ colonnes, telle que $\forall i, j, m_{i,j} = a_{i+j}$ (on numérote en partant de 0).

$$M = \begin{pmatrix} a_0 & a_1 & a_2 & \ldots & a_n \\ a_1 & a_2 & a_3 & \ldots & a_{n+1} \\ a_2 & a_3 & a_4 & \ldots & a_{n+2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n-1} & a_n & a_{n+1} & \ldots & a_{2n-1} \end{pmatrix}$$

Problème de Hankel. Le *problème de Hankel* prend en entrée une suite $(a_0, \ldots, a_{2n-1})$ d'éléments de $\mathbb{K}$. Il demande de trouver un vecteur $v \in \mathbb{K}^{n+1}$ non nul tel que $Mv = 0$, où M est la matrice de Hankel générée par la suite $(a_0, \ldots, a_{2n-1})$.
Problème de réduction polynomiale. Le *problème de réduction polynomiale* prend en entrée une suite $(a_0, \ldots, a_{2n-1})$ d'éléments de $\mathbb{K}$. Il demande de trouver un polynôme $V \in \mathbb{K}[X]$ non nul de degré au plus n tel que $\deg(VA \bmod X^{2n}) < n$, où $A = \sum_{i=0}^{2n-1} a_i X^i \in \mathbb{K}[X]$.

Question 5. Montrer que si on considère le problème de Hankel et le problème de réduction polynomiale tous les deux avec en entrée la même suite $(a_0, \ldots, a_{2n-1})$, alors $v = (v_0, \ldots, v_n)$ est solution du problème de Hankel si et seulement si $V = \sum_{i=0}^{n} v_i X^{n-i}$ est solution du problème de réduction polynomiale.

Polynôme minimal d'une suite. Étant donné une suite $(v_0, \ldots, v_m)$ d'éléments de $\mathbb{K}$, le *polynôme minimal* de cette suite est le polynôme $P = \sum_{i=0}^{d} \lambda_i X^i \in \mathbb{K}[X]$ de degré $d \leq m$ minimal tel que $\forall k \leq m - d, \sum_{i=0}^{d} \lambda_i u_{k+i} = 0$.

Question 6. Supposons qu'on sait résoudre le problème de réduction polynomiale en temps $O(n^2)$. On souhaite maintenant résoudre efficacement le problème de résolution linéaire pour les matrices de faible poids de Hamming. Pour cela, on introduit deux hypothèses simplificatrices.

— M est inversible, et son polynôme minimal est de degré n.
— Si u, v sont deux vecteurs tirés uniformément et indépendamment dans $\mathbb{K}^n$, alors avec probabilité au moins $1/4$, le polynôme minimal de la suite $(u^t M^i v)_{i=0}^{2n-1}$ est de degré n.

a. Montrer qu'avec probabilité au moins $1/4$, le polynôme minimal de la suite $(u^t M^i v)_{i=0}^{2n-1} \in \mathbb{K}^{2n}$ est égal au polynôme minimal de M.
b. En déduire un algorithme qui résout le problème de résolution linéaire pour de telles matrices M en temps $O(n^2 + n\mathsf{H}(M))$ en espérance.

Question 7. Donner un algorithme qui résout le problème de réduction polynomiale en temps $O(n^2)$.

Calcul de l'arbre de dominance

Dans ce sujet, nous nous intéressons à des graphes orientés enracinés, i.e., $G = (V, E, r)$ où V est l'ensemble fini des sommets, E l'ensemble des arêtes orientées, r le sommet racine. Nous supposerons que pour tout sommet $v \in V$ il existe un chemin de r à v, noté $r \to^* v$.
Définition. Étant donné un graphe $G = (V, E, r)$, on dit que $x \in V$ *domine* $y \in V$, noté $x > y$, si, $x \neq y$ et pour tout chemin $r = s_0 \to s_1 \to \ldots \to s_n = y$ dans G, il existe i tel que $s_i = x$.
On dit que x est un *dominant immédiat* de y, noté $idom(y)$, si $x > y$ et pour tout $z \in V$ tel que $z > y$ et $z \neq x$, on a $z > x$.
Définition. Étant donné un graphe $G = (V, E, r)$. On note $A(G) = (V, E_d, r)$ où $E_d = \{x \to y \mid x, y \in V \text{ et } x = idom(y)\}$.
$A(G)$ est appelé l'*arbre de dominance* de G.
L'objectif de ce sujet est de calculer efficacement $A(G)$.

Question 1. Donner l'arbre de dominance du graphe donné en Figure 1.

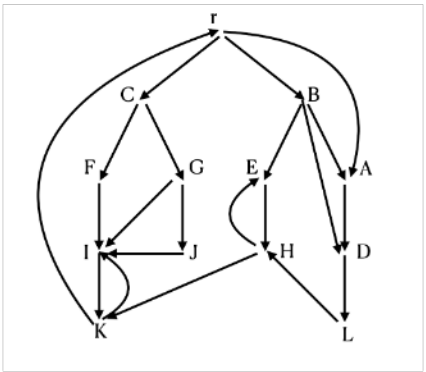


FIGURE 1 – Graphe de la question 1

Question 2. Montrer que $A(G)$ est bien défini, i.e., $idom(x)$ est bien défini pour tout $x \neq r$ et $A(G)$ forme un arbre enraciné en r.

Question 3. a. Donner un algorithme basé uniquement sur des parcours, permettant de calculer $A(G)$.
b. Quelle est sa complexité ?

Soit $G = (V, E, r)$ un graphe. Pour chaque sommet $x \in V$, nous notons $i(x)$ l'indice de visite lors d'un parcours en profondeur issu de r. Nous notons $T(G)$ l'arbre couvrant obtenu à la fin du parcours, i.e., le graphe (V, E_T, r) où E_T est composé des arcs de liaisons durant le parcours.

Définition. Soit $w \neq r$. On note $sdom(w) = argmin_v\{i(v) \mid \text{il existe un chemin } v = v_0 \to \ldots \to v_k = w \text{ dans } G \text{ tel que } i(v_j) > i(w) \text{ pour } 1 \leq j \leq k-1\}$.

Question 4.
a. Donner l'arbre couvrant ainsi que les indices de visite obtenus via un parcours en profondeur depuis r dans le graphe donné en Figure 1.
b. Préciser pour chaque sommet w, le $sdom(w)$ correspondant.

Question 5. Soient $x, y \in V$ tels que $i(x) \leq i(y)$. Montrer que tout chemin $x \to^* y$ dans G contient un sommet a qui est un ancêtre commun à x et y dans $T(G)$.

Question 6. Montrer que pour tout $w \neq r$, $sdom(w)$ est un ancêtre de w dans $T(G)$.

Question 7. Montrer que :

$$sdom(w) = argmin(\quad \{i(v) \mid (v,w) \in E \text{ and } i(v) < i(w)\} \\ \cup \{i(sdom(u)) \mid i(u) > i(w) \text{ and } \exists (v \to w) \in E \text{ tel que } u \to^* v \text{ dans } T(G)\} \ )$$

Axiome. soit $w \neq r$ et $u = argmin_u\{i(sdom(u)) \mid sdom(w) \to^+ u \to^* w$ dans T(G)$\}$. On a

$$idom(w) = \begin{cases} sdom(w) & \text{si } sdom(w) = sdom(u) \\ idom(u) & \text{sinon} \end{cases}$$

Question 8. a. Donner un algorithme qui permet de calculer $idom(w)$ pour tout $w \neq r$.
Conseil : nous pourrons envisager de re-construire $T(G)$ en partant d'une forêt de sommets et en les fusionnant lors d'un parcours des sommets dans un ordre décroissant de leur index.

b. Étudier la complexité de l'algorithme. Nous tiendrons compte de la structure de données utilisée pour manipuler la forêt.

Bob l'écureuil

Bob est un écureuil qui vit le long d'une ligne de chemin de fer. Il a caché ses réserves de noisettes dans les différentes gares, et cherche maintenant à établir son nid dans l'une de ces gares. Pour des raisons pratiques, il souhaite construire son nid dans une gare où il a déjà caché des noisettes, tout en minimisant la durée moyenne entre son nid et ses réserves de noisettes. Comment choisir dans quelle gare établir son nid ?

Question 1. La ligne est composée de 9 gares, numérotées de 0 à 8 ; Bob a mis une cachette dans chacune de ces gares. On met k minutes pour aller de la gare ℓ à la gare $k + \ell$. Quelle gare Bob choisira-t-il d'établir son nid ?

Question 2. On suppose maintenant que les gares ne sont plus régulièrement espacées. Désormais, il y a n gares, toujours numérotées de 0 à $n-1$, toujours en ligne droite. Bob connaît des entiers $t_0, t_1, \ldots, t_{n-2}$ pour lesquels aller de la gare ℓ à la gare $\ell + 1$ requiert t_ℓ minutes ; ainsi, aller de la gare ℓ à la gare $\ell + 2$ requiert $t_\ell + t_{\ell+1}$ minutes.
Donner un algorithme qui permettra à Bob de choisir, en temps $\mathcal{O}(n)$, dans quelle gare installer son nid.

Question 3. Désormais, le réseau prend la forme d'un graphe connexe acyclique. Bob dispose, pour chaque gare k, d'une liste des gares ℓ auxquelles la gare k est reliée directement, ainsi que de la durée $\mathsf{d}_{k,\ell}$, en nombre de minutes, du trajet entre les gares k et ℓ.
Donner un algorithme qui permettra à Bob de choisir, en temps $\mathcal{O}(n)$, dans quelle gare installer son nid.

Bob déménage à Manhattan, là où toutes les rues sont orientées selon un axe Nord-Sud ou Est-Ouest, et espacées de 100 mètres l'une de l'autre. Il a caché ses n réserves de noisettes à n croisements de rue, chaque croisement étant identifié par des coordonnées entières. Par exemple, le croisement $(2, 3)$ se trouve 400 mètres à l'Ouest et 500 mètres au Sud du croisement $(6, 8)$, donc Bob doit marcher au minimum 900 mètres pour aller d'un croisement à l'autre.
Bob souhaite maintenant établir son nid à l'un des n croisements de rue où se trouve déjà une réserve de noisettes, mais tout en minimisant la distance moyenne entre son nid et des réserves de noisettes. À quel croisement établir son nid ?

Question 4. Bob a disposé ses 6 réserves aux points de coordonnées $(0, 0)$, $(1, 1)$, $(2, 3)$, $(3, 1)$, $(3, 2)$ et $(4, 3)$. Auquel de ces six endroits choisira-t-il d'établir son nid ?

Question 5. Bob a simplement noté les coordonnées entières $(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})$ des n réserves qu'il a choisies pour entreposer ses noisettes.
Donner un algorithme qui permettra à Bob de choisir, en temps $\mathcal{O}(n \log(n))$, à quelle intersection installer son nid.

Question 6. On suppose désormais que chacune des coordonnées x_i et y_i est comprise entre 0 et $n^2 - 1$. Comment adapter l'algorithme précédent pour s'assurer qu'il fonctionne désormais en temps $\mathcal{O}(n)$?

Question 7. Pour fluidifier le trafic, des urbanistes ont ajouté des routes orientées du Nord-Est vers le Sud-Ouest, et du Nord-Ouest vers le Sud-Est. Ainsi, le croisement $(2, 3)$ se désormais à $400\sqrt{2} + 100 \approx 666$ mètres du croisement $(6, 8)$: il suffit d'aller quatre fois vers le Nord-Est puis une fois vers le Nord. On suppose toujours que chacune des coordonnées x_i et y_i est comprise entre 0 et $n^2 - 1$, et que Bob veut installer son nid en l'une des n intersections (x_i, y_i).
Donner un algorithme qui permettra à Bob de choisir, en temps $\mathcal{O}(n)$, à quelle intersection installer son nid.

Langages rationnels et lemme de l'étoile

On recherche un langage $\mathcal{L}$ non rationnel mais pour lequel $\mathcal{L}$ et son complémentaire satisfont le lemme de l'étoile.

Question 1. Soit Σ un alphabet fini. Le langage $\mathcal{L}_1$ formé des mots $w \in \Sigma^*$ dont la longueur est paire est-il rationnel ?

Question 2. Le langage $\mathcal{L}_2$ formé des mots $w \in \Sigma^*$ dont la longueur est le carré d'un entier est-il rationnel ?

Soit $\mathcal{L}$ et $\mathcal{L}'$ deux langages sur un alphabet Σ. On dit que $\mathcal{L}$ est $\mathcal{L}'$-*étoilé* s'il existe un entier $n \geqslant 0$ tel que tout mot $w \in \mathcal{L}'$ de longueur $|w| \geqslant n$ admet une factorisation $w = s \cdot t \cdot u$ pour laquelle $|s| \leqslant n$, $1 \leqslant |t| \leqslant n$ et $s \cdot t^* \cdot u \subseteq \mathcal{L} \cup \{w\}$. Si $\mathcal{L}$ est $\mathcal{L}$-étoilé, on dit même que $\mathcal{L}$ satisfait le *lemme de l'étoile*.

Question 3. Démontrer que tout langage rationnel satisfait le lemme de l'étoile.

Question 4. Démontrer que, si $|\Sigma| = 1$, les langages qui satisfont le lemme de l'étoile sont les langages rationnels.

Soit $\mathcal{S}$ une partie finie de Σ^*. On note $\mathcal{M}(\mathcal{S})$ l'ensemble des mots $s_1 \cdot s_2 \cdots s_n$ que l'on peut factoriser en n éléments de $\mathcal{S}$ et pour lesquels il existe deux entiers i et $j = i + 1$ ou $j = i + 2$ tels que $s_i = s_j$.

Question 5. Démontrer que tout ensemble $\mathcal{M}(\mathcal{S})$ est rationnel.

Question 6. Démontrer que, si $|\mathcal{S}| \leqslant 4$, le langage $\mathcal{M}(\mathcal{S})$ est $\mathcal{S}^*$-étoilé.

Question 7. On considère l'alphabet $\Sigma = \{a, b, c, d\}$. Pour tout entier $k \leqslant 3$, on note σ_k le mot $(abc)^k \cdot (abd)^{3-k}$. Les langages $\mathcal{L}_3 = \{(\sigma_0 \cdot \sigma_1 \cdot \sigma_2)^\ell \cdot (\sigma_0 \cdot \sigma_1 \cdot \sigma_3)^\ell : \ell \geqslant 0\}$ et $\mathcal{L} = \mathcal{M}(\{\sigma_0, \sigma_1, \sigma_2, \sigma_3\}) \cup \mathcal{L}_3$ sont-ils rationnels ?

Question 8. Démontrer que les ensembles $\mathcal{L}$ et $\Sigma^* \setminus \mathcal{L}$ satisfont le lemme de l'étoile.

Algorithme Union-Find

L'algorithme Union-Find a pour but de gérer dynamiquement une partition d'un ensemble $\{1, 2, \ldots, n\}$ fixé. Initialement, on part de la partition maximale $\{\{1\}, \{2\}, \ldots, \{n\}\}$. En pratique, on représente les partitions de $\{1, \ldots, n\}$ comme suit :

- ▷ chaque entier k possède un (seul) *père* $\mathsf{p}_k \leqslant n$, ce que l'on notera $k \to \mathsf{p}_k$; on peut avoir $k = \mathsf{p}_k$;
- ▷ chaque entier k possède un *poids* $\mathsf{w}_k \in \mathbb{N}$;
- ▷ deux entiers distincts k et ℓ ne peuvent être ancêtres l'un de l'autre, et appartiennent au même sous-ensemble si et seulement s'ils ont un ancêtre commun ;
- ▷ si l'entier k appartient à un singleton, $w_k = 0$.

Question 1. On dit que m est le *chef* de k si m est un ancêtre de k tel que $m = \mathsf{p}_m$. Démontrer que tout entier a un unique chef, et deux entiers k et ℓ appartiennent au même sous-ensemble si et seulement s'ils ont le même chef.

L'algorithme est censé gérer trois types de requêtes, en procédant comme suit :

1. La requête auxiliaire $\mathsf{Chef}(k)$: on recherche le chef de k. Si $k = \mathsf{p}_k$, il s'agit de k lui-même. Sinon, il s'agit du chef de p_k, et ce chef devient le nouveau parent de k.
2. La requête $\mathsf{Test}(k, \ell)$: on se demande si k et ℓ appartiennent au même sous-ensemble. Cela revient à identifier les entiers $k' = \mathsf{Chef}(k)$ et $\ell' = \mathsf{Chef}(\ell)$ puis à vérifier si $k' = \ell'$.
3. La requête $\mathsf{Union}(k, \ell)$: on souhaite réunir les sous-ensembles auxquels appartiennent k et ℓ. Cela revient à identifier les entiers $k' = \mathsf{Chef}(k)$ et $\ell' = \mathsf{Chef}(\ell)$, puis :
   - ▷ si $\mathsf{w}_{k'} > \mathsf{w}_{\ell'}$, l'entier k' devient le nouveau parent de ℓ' ;
   - ▷ si $\mathsf{w}_{\ell'} > \mathsf{w}_{k'}$, l'entier ℓ' devient le nouveau parent de k' ;
   - ▷ si $k' \neq \ell'$ et $\mathsf{w}_{k'} = \mathsf{w}_{\ell'}$, l'entier k' devient le nouveau parent de ℓ' et son poids augmente de 1.

On souhaite démontrer que répondre à m de ces requêtes peut se faire en temps $\mathcal{O}(m \log^\star(m))$, où $\log^\star$ est la fonction définie par $\log^\star(m) = 1$ lorsque $m \leqslant 1$ et $\log^\star(m) = 1 + \log^\star(\log_2(m))$ lorsque $m > 1$.

Question 2. En partant de l'ensemble $\{\{1\}, \{2\}, \{3\}, \{4\}\}$, on effectue successivement les requêtes $\mathsf{Union}(1, 2)$, $\mathsf{Union}(3, 4)$, $\mathsf{Union}(2, 4)$ et $\mathsf{Test}(2, 4)$. Indiquer le père et le poids de chaque entier $k \leqslant 4$.

Question 3. On note w_k^∞ le poids d'un entier k à la fin de l'algorithme. Démontrer que $\mathsf{w}_u^\infty < \mathsf{w}_v^\infty$ pour tous les entiers $u \neq v$ tels que v a été le parent de u.

Question 4. Démontrer, pour tout entier $\ell \geqslant 1$, qu'il y a au plus $m/2^{\ell-1}$ entiers $k \leqslant n$ pour lesquels $\mathsf{w}_k^\infty = \ell$.

Question 5. Démontrer que toute requête Test ou Union effectuée au cours de l'algorithme a une complexité majorée par $\mathcal{O}(\log_2(\min\{m, n\}))$.

Question 6. Soit $\mathcal{G} = (V, E)$ le graphe dont les sommets sont les entiers de 1 à n et les arêtes sont les paires (u, v) pour lesquelles $u \neq v$ et v a été le parent de u au cours de l'algorithme, mais pas quand celui-ci se termine. Démontrer que la complexité totale de nos m requêtes est majorée par $\mathcal{O}(m + |E|)$.

Question 7. Soit $(a_\ell)_{\ell \geqslant 0}$ la suite définie par $a_0 = 0$ et $a_{\ell+1} = 2^{a_\ell}$. Pour tout entier $\ell \geqslant 0$, on note V_ℓ l'ensemble des entiers k tels que $a_\ell \leqslant \mathsf{w}_k^\infty < a_{\ell+1}$. Démontrer que $\mathcal{G}$ contient au plus $4m$ arêtes reliant deux sommets de V_ℓ, et au plus $2m$ arêtes allant d'un sommet de V_ℓ à un sommet en dehors de V_ℓ.

Question 8. Que conclure sur la complexité de l'algorithme Union-Find ?

Codage par couleurs

On considère des graphes orientés, supposés sans boucle. On cherchera dans ces graphes un chemin d'une longueur spécifique (sans imposer de point de départ et d'arrivée), où la *longueur* d'un chemin est le nombre de sommets qui apparaissent dans le chemin. Sauf mention explicite du contraire, on considère toujours des chemins *simples*, c'est-à-dire qui ne passent pas deux fois par le même sommet.

Question 1. Y a-t-il des graphes fortement connexes arbitrairement grands sans chemin de longueur 4 ?

Question 2. Proposer un algorithme naïf pour déterminer, étant donné un graphe G et un entier $k \in \mathbb{N}$, si G contient un chemin de longueur k. Quelle est la complexité de cet algorithme ?

Question 3. Dans cette question seulement, on s'intéresse à des chemins *non nécessairement simples*. Proposer un algorithme efficace pour déterminer, étant donné un graphe G et un entier $k \in \mathbb{N}$, si G contient un tel chemin de longueur k.

Question 4. Dans cette question seulement, on suppose que les sommets du graphe d'entrée G portent chacun une couleur parmi l'ensemble $\{1, \ldots, k\}$, et on souhaite savoir si G admet un chemin *multicolore* de longueur k, c'est-à-dire un chemin $v_1, \ldots, v_k$ où les couleurs des sommets sont deux à deux distinctes.
Proposer un algorithme pour résoudre ce problème, et en expliciter la complexité.

Pour améliorer le temps d'exécution de l'algorithme de la question 2, on va concevoir un algorithme *probabiliste*. Un tel algorithme a la possibilité de tirer au hasard certaines valeurs au cours de son exécution ; et on regarde, sur chaque entrée, quelle est la probabilité que l'algorithme réponde correctement, en fonction de ces tirages aléatoires.
On veut résoudre le problème suivant : étant donné un graphe G et un entier k, on veut savoir si G a un chemin de longueur k. On considère d'abord l'algorithme (1) que voici. On répète M fois l'opération suivante (où M sera déterminé ensuite) : tirer k sommets au hasard et vérifier si ces sommets forment un chemin. On répond OUI si l'un de ces tirages est réussi et NON dans le cas contraire.

Question 5. On suppose que le graphe G a n sommets et contient précisément $1 \leqslant c \leqslant n^k$ chemins de longueur k. Exprimer la probabilité que l'algorithme (1) réponde OUI, en fonction de M, de k, de n, et de c.

Question 6. Si on suppose que G n'a pas de chemin de longueur k, que répond l'algorithme (1) ?

Question 7. Pour $M = n^k$, montrer que l'algorithme répond correctement dans les deux cas avec probabilité au moins $1/2$.

Question 8. Quel est le temps d'exécution de l'algorithme (1) pour ce choix de M ? Commenter.

On considère à présent l'algorithme (2) dont le principe est le suivant. On répète M fois l'opération suivante (où M sera déterminé ensuite) : tirer un ordre total aléatoire $v_1 < \cdots < v_n$ sur les sommets de G, retirer les arêtes (u, v) où on a $u > v$, et vérifier si le graphe résultant $G_<$ a un chemin de longueur k (d'une manière qui reste à déterminer).

Question 9. Quelle propriété ont les graphes $G_<$ construits par cet algorithme ? Comment exploiter cette propriété pour trouver efficacement les chemins de longueur k ?

Question 10. Proposer un M qui garantisse que cet algorithme réponde correctement avec une probabilité $\geqslant 1/2$. Quelle complexité obtient-on ? Commenter.

Question 11. En utilisant les chemins multicolores, proposer un algorithme probabiliste qui répond correctement avec probabilité au moins $1/2$ et s'exécute en $O((2e)^k \times (n + m))$ sur un graphe à n sommets et m arêtes.

Algorithme du tri lent

On souhaite trier un tableau A de longueur n en utilisant l'algorithme de TriLent présenté ci-dessous. L'objectif de ce problème est de démontrer que l'algorithme est correct et d'évaluer certaines statistiques liées à sa complexité.

```
1 Fonction TriLent(A_{1...n}):
2   pour i = 1, 2, ..., n :
3     pour j = 1, 2, ..., n :
4       si A_i < A_j : échanger A_i et A_j
```

Question 1. Quelle est, en fonction de n, la complexité de l'algorithme de TriLent, en nombre de comparaisons, dans le pire cas ? dans le meilleur cas ?

Question 2. Un algorithme de tri est *stable* si deux objets A_i et A_j égaux pour notre fonction de comparaison et pour lesquels $i < j$ sont envoyés en deux positions u et v telles que $u < v$. Le TriLent est-il stable ?

Question 3. Soit i un entier tel que $1 \leqslant i \leqslant n$. Démontrer que, juste après avoir effectué i fois la boucle **pour** des lignes 3 et 4, l'élément A_i est l'élément maximal du tableau, et le sous-tableau formé des entrées $A_1, A_2, \ldots, A_i$ est trié dans l'ordre croissant.

Question 4. Démontrer que le TriLent permet effectivement de trier le tableau fourni en entrée.

Question 5. On suppose que les n éléments du tableau fourni en entrée sont deux à deux distincts. Donner le plus petit nombre possible d'échanges que le TriLent peut être amené à effectuer.

Question 6. Le tableau fourni en entrée est une permutation des entiers $1, 2, \ldots, n$ choisie uniformément au hasard. Démontrer que le nombre moyen d'échanges qu'effectuera le TriLent est égal à $n(n-1)/4 + 2H_n - 2$, où

$$H_n = \frac{1}{1} + \frac{1}{2} + \cdots + \frac{1}{n}$$

est le $n^{\text{ème}}$ nombre harmonique.

Question 7. Donner le nombre maximal d'échanges que l'algorithme de TriLent peut être amené à effectuer.

Éviter les carrés abéliens

Étant donné un mot w sur un alphabet $\Sigma = \mathbb{Z}/k\mathbb{Z}$, on note $|w|$ la longueur et $|w|_x$ le nombre d'occurrences d'une lettre $x \in \Sigma$ dans le mot w. Deux mots w et w' sont des *jumeaux abéliens*, ce que l'on note $w \equiv w'$, si $|w|_x = |w'|_x$ pour toute lettre $x \in \Sigma$. Enfin, un mot w est un *carré abélien* s'il admet une factorisation $w = uv$ telle que $u \equiv v$, et w *évite* les carrés abéliens si aucun de ses facteurs non vides n'est un carré abélien.

Question 1. Parmi les mots ci-dessous, lesquels sont des carrés abéliens ? Lesquels évitent les carrés abéliens ?

102120 012330 0123012 102040203040301

Question 2. Démontrer que nul mot de longueur 8 sur l'alphabet $\mathbb{Z}/3\mathbb{Z}$ n'évite les carrés abéliens.

Ci-dessous, on souhaite démontrer qu'il existe des mots arbitrairement longs, sur l'alphabet $\Sigma = \mathbb{Z}/5\mathbb{Z}$, qui évitent les carrés abéliens. Dans ce but, on note $\mathbf{W}$ le dernier des quatre mots donnés en question 1, puis $\sigma : \Sigma^* \to \Sigma^*$ et $\phi : \Sigma^* \to \Sigma^*$ les morphismes de monoïdes définis par $\sigma : k \mapsto k+1$ et $\phi : k \mapsto \sigma^k(\mathbf{W})$. On suppose alors qu'il existe un mot w, aussi court que possible, évitant les carrés abéliens et tel que $\phi(w)$ n'évite pas les carrés abéliens ; cette supposition se révèlera incorrecte.

Question 3. Comment, à l'aide d'un ordinateur, s'assurer que $|w| \geqslant 3$?

On dit que neuf mots $(k, \ell, m, p, q, r, s, t, u)$ forment un *beau* nonuplet si (1) chacun des mots k, ℓ et m est vide ou réduit à une lettre ; (2) p est un suffixe de $\phi(k)$, rs est une factorisation de $\phi(\ell)$, u est un préfixe de $\phi(m)$; (3) chacun des mots p, r, s et u est de longueur au plus 14 ; et (4) $p\phi(q)r \equiv s\phi(t)u$, tandis que $q \not\equiv \ell t$, $q \not\equiv \ell tm$, $kq \not\equiv \ell t$, $kq \not\equiv \ell tm$, $q\ell \not\equiv t$, $q\ell \not\equiv tm$, $kq\ell \not\equiv t$ et $kq\ell \not\equiv tm$.

Question 4. Pourquoi existe-t-il nécessairement un beau nonuplet $(k, \ell, m, p, q, r, s, t, u)$ tel que $w = kq\ell tm$?

Question 5. Démontrer que $|q| - 1 \leqslant |t| \leqslant |q| + 1$.

Question 6. Démontrer que $|p|_x + 2|q| + 5|q|_x + |r|_x = |s|_x + 2|t| + 5|t|_x + |u|_x$ pour toute lettre $x \in \Sigma$.

Question 7. Comment, à l'aide d'un ordinateur, s'assurer qu'il n'existe aucun beau nonuplet ?

Question 8. On suppose les vérifications des questions 3 et 7 effectuées [1]. Démontrer, pour tout entier $n \geqslant 0$, qu'il existe un mot w de longueur n sur l'alphabet $\Sigma = \mathbb{Z}/5\mathbb{Z}$ qui évite les carrés abéliens.

1. Cela a pris cinq secondes sur l'ordinateur des examinateurs.

Nœuds importants d'un arbre

On considère un arbre à n nœuds. L'*excentricité* d'un nœud est sa distance maximale à un autre nœud de l'arbre. Le *diamètre* d'un arbre est le plus long chemin entre deux nœuds de l'arbre. Le *rayon* d'un arbre est la plus petite excentricité d'un nœud.

Question 1. Donner un algorithme pour déterminer l'excentricité d'un nœud, et sa complexité.

Question 2. Montrer qu'il y a toujours au plus deux nœuds d'excentricité minimale.

Question 3. Donner un algorithme qui identifie efficacement un tel nœud d'excentricité minimale.

Question 4. Montrer que le rayon r d'un arbre et le diamètre D d'un arbre vérifient $r = \lceil D/2 \rceil$.

Question 5. Donner un algorithme efficace pour identifier le plus long chemin dans un arbre.

Question 6. On imagine que ces nœuds représentent des ordinateurs connectés entre eux et on doit choisir un unique nœud qu'on va relier à Internet, qui partagera sa connexion avec sa composante connexe. En cas de coupure d'une unique arête de l'arbre, on souhaite minimiser le nombre d'ordinateurs privés d'Internet. Donner une méthode qui identifie le nœud à connecter à Internet dans le pire cas.

Question 7. On suppose maintenant l'arbre pondéré par des poids entiers w_{uv} entre u et v. On s'intéresse à colorier k nœuds de façon que si $d(v)$ est le plus court chemin du nœud v à un nœud colorié, la valeur maximale de d sur le graphe soit aussi petite que possible. On demande de renvoyer cette distance dans le pire cas.